Abstract: Software Systems are evolving by adding new functions and modifying existing functions over time. Through the evolution process, copy paste programming and other processes leads to duplication of data resulting in model clones or code clones. Since clones are believed to reduce the maintainability of software. Several code clone detection techniques and tools have been proposed. This paper proposes a new clone detection technique to outwit the hindrance of clones by applying a 3-way approach of detecting and removing the clones. The 3-way approach for cloning integrates the three aspects of software engineering: Model Based Visual Analysis, Pattern Based Semantic Analysis and Syntactical Code Analysis. The process is automated by developing a tool that requires no parsing yet is able to detect a significant amount of code duplication. The existence of replicated code in a system makes that system harder to maintain and evolve. To remove replicated code the usual way is to use refactoring. However there are always clones that cannot be removed by refactoring alone. Some are due to lack of composition mechanisms in the underlying programming language. We propose the use of roles to remove such clones since roles provide a finer degree of composition. We sketch four role refactoring to remove code clones and apply them in a case study using the Outdraw framework. Results show that roles have a positive impact in clone reduction as they were able to remove almost all clones traditional refactoring could not.

I. INTRODUCTION
Software Clones affects software maintenance and other engineering activities. Hence clone detection and removal has grown as an active area in software engineering research community yielding numerous techniques, various tools and other methods for clone detection and removal. From requirement analysis till software maintenance, it is important to consider various factors that affect its quality, reuse and maintenance. In this respect, software cloning plays an important role. Detecting and removing clones and redundant data will improve the overall efficiency of the software and specifically will ease the maintenance and reuse of the components from the repositories. Cloning works at the cost of increasing lines of code without adding to overall productivity. Same software bugs and defects are replicated that reoccurs throughout the software at its evolving as well its maintenance phase. It results to excessive maintenance costs as well. So cut paste programming form of software reuse raise the number of lines of code without expected reduction in maintenance costs associated with other forms of reuse. So, to eliminate code clones, is a promising way to reduce the maintenance cost in future. In this paper three different aspects of software engineering are considered and are integrated to detect and possibly remove the clones. First, Model based Visual Analysis. Model clones are segments of models that are similar according to some definition of similarity. During software development complete system is modeled by UML diagrams. The model clones can be detected within UML diagrams at the initial phase of development. The removal of model clones will prevent further penetration of model clones as code clones. Second aspect is pattern based semantic analysis. Refactoring patterns are used to find the cloned codes. Refactoring pattern improves the code by detecting clones and then removing them. Here four cloning patterns are discussed namely extract, pull up, template and strategy pattern. If the same code structure occurs in more than one place, it’s sure that code will become better if it is unified. The simplest duplicated code problem is having the same expression in two methods of the same class. Here Extract pattern can be used and the code is invoked from both places. Similarly Pull Up, Template and Strategy patterns are used to detect and remove the unnecessary code clones. Also if all the cloned fragments of a same source fragment can be detected, the functional usage patterns of that fragment can be discovered. Third one is syntactical code analysis. Clones can be detected based on the degree of similarity that varies from an exact superficial copy to some near miss similar regions of code or structurally similar regions of code. So, we need a better approach for the analysis of clone detection and its removal. Also some cloning concepts need to be revised and flaws need to be eliminated. An approach that could integrate cloning at or before design as well as code level for the effective maintenance is needed. An automated system or tool is required, that can automatically detect and remove clones. This will definitely contribute to reduce the maintenance efforts. This work contributes a
methodology by integrating Model-Based Visual Analysis, Pattern Based Semantic Analysis and Syntactical Analysis of Code for better detection of clones not only at code level but also at other levels of software development. And it thereby improves the code, design, quality and maintenance of software as a whole. Over the last decade many techniques and tools for software clone detection have been proposed. In this paper, we provide a qualitative comparison and evaluation of the current state-of-the-art in clone detection techniques and tools, and organize the large amount of information into a coherent conceptual framework. We begin with background concepts, a generic clone detection process and an overall taxonomy of current techniques and tools. We then classify, compare and evaluate the techniques and tools in two different dimensions. First, we classify and compare approaches based on a number of facets, each of which has a set of (possibly overlapping) attributes. Second, we qualitatively evaluate the classified techniques and tools with respect to a taxonomy of editing scenarios designed to model the creation of Type-1, Type-2, Type-3 and Type-4 clones. Finally, we provide examples of how one might use the results of this study to choose the most appropriate clone detection tool or technique in the context of a particular set of goals and constraints. The primary contributions of this paper are: (1) a schema for classifying clone detection techniques and tools and a classification of current clone detectors based on this schema, and (2) a taxonomy of editing scenarios that produce different clone types and a qualitative evaluation of current clone detectors based on this taxonomy. Reusing code fragments by copying and pasting with or without minor adaptation is a common activity in software development. As a result software systems often contain sections of code that are very similar, called code clones. Previous research shows that a significant fraction (between 7% and 23%) of the code in a typical software system has been cloned. While such cloning is often intentional and can be useful in many ways, it can also be harmful in software maintenance and evolution. For example, if a bug is detected in a code fragment, all fragments similar to it should be checked for the same bug. Duplicated fragments can also significantly increase the work to be done when enhancing or adapting code. Many other software engineering tasks, such as program understanding (clones may carry domain knowledge), code quality analysis (fewer clones may mean better quality code), aspect mining (clones may indicate the presence of an aspect), plagiarism detection, copyright infringement investigation, software evolution analysis, code compaction (for example, in mobile devices), virus detection, and bug detection may require the extraction of syntactically or semantically similar code fragments, making clone detection an important and valuable part of software analysis Definition 3 (Clone Types). There are two main kinds of similarity between code fragments. Fragments can be similar based on the similarity of their program text, or they can be similar based on their functionality (independent of their text). The first kind of clone is often the result of copying a code fragment and pasting into another location. In the following we provide the types of clones based on both the textual (Types 1 to 3) and functional (Type 4) similarities: Type-1: Identical code fragments except for variations in whitespace, layout and comments. Type-2: Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments. Type-3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments. Type-4: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

II. THEORETICAL FRAME WORKS

A) Metrics-Based Data Mining Approach for Software Clone Detection

Metrics-Based Data Mining Approach for Software Clone Detection (2012 IEEE 36th International Conference on Computer Software and Applications)

States that the detection of function clones in software systems is valuable for the code adaptation and error checking maintenance activities. This paper presents an efficient metrics-based data mining clone detection approach. First, metrics are collected for all functions in the software system. A data mining algorithm, fractal clustering, is then utilized to partition the software system into a relatively small number of clusters. Each of the resulting clusters encapsulates functions that are within a specific proximity of each other in the metrics space. Finally, clone classes, rather than pairs, are easily extracted from the resulting clusters. For large software systems, the approach is very space efficient and linear in the size of the data set. Evaluation is performed using medium and large open source software systems. In this evaluation, the effect of the chosen metrics on the detection precision is investigated.

B) Survey on Software Clone Detection Research

Survey on Software Clone Detection Research (Chanchal Kumar Roy and James R. Cordy September 26, 2007) describes about the Code duplication or copying a code fragment and then reuse by pasting with or without any modifications is a well known code smell in software maintenance. Several studies show that about 5% to 20% of software systems can contain duplicated code, which is basically the result of copying existing code fragments and using them by pasting with
or without minor modifications. One of the major shortcomings of such duplicated fragments is that if a bug is detected in a code fragment; all the other fragments similar to it should be investigated to check the possible existence of the same bug in the similar fragments. Refactoring of the duplicated code is another prime issue in software maintenance although several studies claim that refactoring of certain clones are not desirable and there is a risk of removing them. However, it is also widely agreed that clones should at least be detected.

C) Implementing a 3-Way Approach of Clone Detection and Removal using PC Detector Tool

Implementing a 3-Way Approach of Clone Detection and Removal using PC Detector Tool (2014 IEEE) states that the Software Systems are evolving by adding new functions and modifying existing functions over time. Through the evolution process, copy paste programming and other processes leads to duplication of data resulting in model clones or code clones. Since clones are believed to reduce the maintainability of software, several code clone detection techniques and tools have been proposed. This paper proposes a new clone detection technique to outwit the hindrance of clones by applying a 3-way approach of detecting and removing the clones. The 3-way approach for cloning integrates the three aspects of software engineering: Model Based Visual Analysis, Pattern Based Semantic Analysis and Syntactical Code Analysis. The process is automated by developing a tool that requires no parsing yet is able to detect a significant amount of code duplication.

### III. COMPARISON OF THREE PAPERS

<table>
<thead>
<tr>
<th>Technique</th>
<th>Procedure</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper 1</td>
<td>It is Based on data mining based code clone detection approach</td>
<td>It is based on subdividing the larger code into smaller parts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it can increase the complexity as well as use full for identifying type-1 and type-2 clones., it goes for Text comparison</td>
</tr>
<tr>
<td>Paper 2</td>
<td>It is based on refactoring of the duplicated code</td>
<td>Collecting batches of code clones. is another prime issue in software maintenance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It again identifies the code clones by text comparison. This type of technique is use full for type-1 and type-2 clones.</td>
</tr>
<tr>
<td>Paper 3</td>
<td>3-way approach for cloning integrates the three aspects of software engineering</td>
<td>It is model base visual analysis pattern based semantic analysis and syntactical code analysis.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It only checks the code clones of type-1 and type-2 type of clones.</td>
</tr>
<tr>
<td>Our Research</td>
<td>Till now code clone has been detected till type-2 type of clones. In our research we can improve the type-2 clones and also detect the type-3 clones. We can use the technique of PDG. in latest paper of 3-ways approach we can consider the base.</td>
<td></td>
</tr>
</tbody>
</table>

### IV. CONCLUSION

In our research work we will be able to improve the coding procedure. Because now a day’s code are so enlarged. There requires large amount of debugging time. It may increase the cost to software to substantial levels. But using code clone detection and removal we will be able to identify the code clones and there common procedures can be developed. Those common procedures can be called rather than recode again and again. It will remove the storage space requirements and debugging time to substantial levels.

### V. FUTURE WORK

In our research work we are considering 3-way approach to identify the code clone. In base paper they have developed the tool to identify type 1 and type 2 clones. Now we will enhance this tool to incorporate Type-3 type of clones. But in our research work we will not be able to identify the type-4 type of clones. That can be done in future course of actions.
REFERENCES


